Rozdział 23
PDO

Wprowadzanie do PDO, kursory zwykłe, kursory przewijane, obsługa błędów, przetwarzanie transakcyjne

Wprowadzenie

- **PDO – PHP Data Objects,**
- **obiektowo-zorientowany interfejs programistyczny dostępu do SQL-owych baz danych z poziomu PHP,**
- **lekki:**
  - **podstawowe API**
  - **wydajny ponieważ zakodowany w C**
- **generyczny interfejs niezależny od źródła danych, zapewnia współpracę aplikacji z różnymi bazami danych:**
  - **FreeTDS / Microsoft SQL Server / Sybase**
  - **Firebird/Interbase 6**
  - **IBM DB2**
  - **IBM Informix Dynamic Server**
  - **MySQL 3.x/4.x**
  - **Oracle Call Interface**
  - **ODBC v3 (IBM DB2, unixODBC and win32 ODBC)**
  - **PostgreSQL**
  - **SQLite 3 and SQLite 2**

Podstawowe kroki aplikacji

1. **dołączenie do źródła danych**
2. **określenie parametrów połączenia**
3. **przygotowanie i wykonanie polecenia**
4. **pobranie wyników**
5. **odłączenie od źródła danych (opcja)**

Źródło danych

- **`PDO->__construct()` – konstruktor klasy PDO ustanawia połączenia ze źródłem danych,**
  - **parametry:**
    - źródło danych (DSN),
    - nazwa użytkownika (opcja),
    - hasło użytkownika (opcja),
    - tablica z parametrami (opcja),
  - **wyjście: obiekt klasy PDO reprezentujący połączenie do bazy danych**
- **zamknięcie połączenia:**
  - **utrata zakresu zmiennej reprezentującej połączenie,**
  - **podstawienie pod zmienną wartości null**

```
<?php
//Ustanowienie połączenia
$dbh = new PDO('oci:dbname=//dblab.cs.put.poznan.pl/dblab10g.cs.put.poznan.pl',
        'scott', 'tiger');
...
//Zamknięcie połączenia
$dbh=null;
?>
```

Określenie parametrów połączenia

- **`PDO->setAttribute()`– ustawienie atrybutów połączenia:**
  - **adresat:**
    - obiekt klasy PDO reprezentujący połączenie z bazą danych
  - **parametry:**
    - atrybut,
    - wartość,
  - **wyjście:**
    - TRUE w przypadku sukcesu, FALSE w przypadku porażki
  - **przykład:** nazwy kolumn w zbiorze wynikowym wielkimi literami:

```
...
$dbh->setAttribute(PDO::ATTR_CASE, PDO::CASE_UPPER);
...
```


Bezpośrednie wykonanie polecenia

- **odpowiednie dla poleceń wykonywanych w aplikacji jednokrotnie,**
- **polecenie parsowane przy każdym uruchomieniu,**
- **można użyć do wykonania zapytania, jednak nie ma dostępu do jego wyniku (brak zbioru wynikowego),**
- **`PDO->exec()` – bezpośrednie wykonanie polecenia:**
  - **adresat:**
    - obiekt klasy PDO reprezentujący połączenie z bazą danych
  - **parametry:**
    - tekst polecenia SQL,
  - **wyjście:**
    - liczba przetworzonych wierszy

```
...
$dbh->exec("update pracownicy set placa_pod=placa_pod*1.1");
...
```


Bezpośrednie wykonanie zapytania

- **`PDO->query()` – bezpośrednie wykonanie zapytania:**
  - **adresat:**
    - obiekt klasy PDO reprezentujący połączenie z bazą danych
  - **parametry:**
    - tekst polecenia SQL,
  - **wyjście:**
    - obiekt klasy `PDOStatement` reprezentujący wynik zapytania
- **Niepobranie całego wyniku przed ponownym wykonaniem metody `PDO->query()` wymaga zamknięcia kursora za pomocą metody `PDOStatement->closeCursor()`**

```
...
$stmt=$dbh->query("select * from pracownicy");
...
```


Wykonanie przygotowanego polecenia

- **`PDO->prepare()` – przygotowanie polecenia:**
  - **adresat:**
    - obiekt klasy PDO reprezentujący połączenie z bazą danych
  - **parametry:**
    - tekst polecenia SQL,
    - tablica z parametrami dla driver-a (opcja)
  - **wyjście:**
    - obiekt klasy `PDOStatement` reprezentujący przygotowane polecenie
- **`PDOStatement->execute()` – wykonanie przygotowanego polecenia:**
  - **adresat:**
    - obiekt klasy `PDOStatement` reprezentujący przygotowane polecenie
  - **parametry:**
    - tablica ze zmiennymi wiązania dla parametrów wejściowych (opcja)
  - **wyjście:**
    - TRUE w przypadku sukcesu, FALSE w przypadku porażki

```
...
$stmt=$dbh->prepare("select * from pracownicy");
$stmt->execute();
...
```

Detekcja błędów (1)

- **Trzy alternatywne tryby obsługi błędów:**
  - **PDO::ERRMODE_SILENT – tryb domyślny, PDO ustawia kod błędu, który można odczytać za pomocą:**
    - **`PDO->errorCode()`, `PDOStatement->errorCode()` – zwraca SQLCODE - pięcioznakowy, alfanumeryczny ciąg wg standardu ANSI SQL-92**
    - **`PDO->errorInfo()`, `PDOStatement->errorInfo()` – zwraca tablicę trzyelementową:**
      - **1-szy element – SQLCODE**
      - **2-gi element – numer błędu specyficzny dla źródła danych**
      - **3-ci element – tekst błędu specyficzny dla źródła danych**
  - **PDO::ERRMODE_WARNING – oprócz ustawienia kodu błędu, PDO emituje tradycyjny komunikat E_WARNING, przydatny przy testowaniu aplikacji**
  - **PDO::ERRMODE_EXCEPTION – oprócz ustawienia kodu błędu, PDO zgłasza wyjątek PDOException, ułatwia czytelną organizację kodu obsługującego błędy**

Detekcja błędów (2)

```
try {
...
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
...
$dbh->exec("update pracownicy set placa_pod=placa_pod*1.1");
...
} catch (PDOException $e) {
        echo "Błąd wykonania: " . $e->getMessage();
}
```


Zmienne wiązania (1)

- **polecenie przygotowywane może być sparametryzowane,**
- **oznaczenie zmiennej – znak ? lub :nazwa_zmiennej**
- **`PDOStatement->bindParam()` – wiąże zmienną wiązania ze zmienną aplikacji:**
  - **adresat:**
    - obiekt klasy `PDOStatement` reprezentujący przygotowane polecenie
  - **parametry:**
    - pozycja zmiennej lub jej nazwa
    - zmienna aplikacji
    - typ danych dla zmiennej + tryb IN/OUT (opcja):
      - PDO::PARAM_BOOL – reprezentuje typ logiczny,
      - PDO::PARAM_NULL – reprezentuje wartość pustą,
      - PDO::PARAM_INT – reprezentuje typ SQL INTEGER,
      - PDO::PARAM_STR – reprezentuje typy SQL CHAR, SQL VARCHAR,
      - PDO::PARAM_LOB – reprezentuje duży obiekt (LOB),
      - PDO::PARAM_STMT – reprezentuje zbiór wynikowy, aktualnie nie wspierany przez żaden driver,
      - PDO::PARAM_INPUT_OUTPUT – określa jako parametr rodzaju INOUT dla składowanych procedur. Należy użyć bitwise-OR tej wartości i typu danych PDO::PARAM_*.
    - długość zmiennej typu OUT
  - **wyjście:**
    - TRUE w przypadku sukcesu, FALSE w przypadku porażki

Zmienne wiązania (2)

```
...
$stmt=$dbh->prepare('select nazwisko from pracownicy
                where id_zesp=:id_zesp and etat=:etat');
$id_zesp=10; $etat='DYREKTOR';
$stmt->bindParam(':id_zesp', $id_zesp);
$stmt->bindParam(':etat', $etat);
$stmt->execute();
...
$id_zesp=20; $etat='ASYSTENT';
$stmt->execute();

...
```

```
...
$stmt=$dbh->prepare('select nazwisko from pracownicy where id_zesp=? and
etat=?');
$id_zesp=10; $etat='DYREKTOR';
$stmt->bindParam(1, $id_zesp);
$stmt->bindParam(2, $etat);
$stmt->execute();
...
$id_zesp=20; $etat='ASYSTENT';
$stmt->execute();
...
```

Zbiór wynikowy

- **`PDOStatement->columnCount()`– metoda odczytująca liczbę kolumn w zbiorze wynikowym**
  - **adresat:**
    - obiekt klasy `PDOStatement` reprezentujący przygotowane polecenie
  - **wyjście:**
    - liczba kolumn zbioru wynikowego, 0 jeżeli polecenie nie zwraca zbioru wynikowego
- **`PDOStatement->getColumnMeta()`– metoda odczytująca metadane kolumny**
  - **adresat:**
    - obiekt klasy `PDOStatement` reprezentujący przygotowane polecenie
  - **parametry:**
    - numer kolumny (zliczany od zera)
  - **wyjście:**
    - tablica asocjacyjna zawierająca następujące informacje: natywny (PHP) typ danych kolumny wykorzystany do reprezentacji danych z kolumny, typ SQL, flagi ustawione dla kolumny, nazwa kolumny, długość, precyzja, typ PDO

```
<?php
$stmt = $dbh->query('SELECT * FROM pracownicy');
for ($i=0; $i<$stmt->columnCount(); $i++){$meta = $stmt->getColumnMeta($i);
var_dump($meta);}
?>
```


Przeglądanie zbioru wynikowego (1)

- **`PDOStatement->fetch()`– metoda pobierająca kolejny wiersz zbioru wynikowego**
  - **adresat:**
    - obiekt klasy `PDOStatement` reprezentujący przygotowane i wykonane polecenie
  - **parametry:**
    - stała określająca typ wartości zwrotnej
  - **wyjście:**
    - wartość zwrotna, której typ jest zależny od wartości parametru:
      - PDO::FETCH_NUM – tablica indeksowana numerem pozycji kolumny w zbiorze wynikowym (od 0), w każdym elemencie tablicy znajduje się kolejna wartość kolumny pobranego wiersza

```
<?php
while ($row = $stmt->fetch(PDO::FETCH_NUM)) {
        printf("Nazwisko %s, etat %s<br>", $row[0], $row[1]); }
?>
```

      - PDO::FETCH_ASSOC – tablica asocjacyjna indeksowana nazwami kolumn

```
<?php
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        echo "Nazwisko $row[NAZWISKO], etat $row[ETAT]<br>"; }
?>
```


Przeglądanie zbioru wynikowego (2)

      - PDO::BOTH – tablica indeksowana zarówno pozycjami jak i nazwami kolumn
      - PDO::FETCH_OBJ – obiekt anonimowy, którego nazwy własności odpowiadają nazwom kolumn

```
<?php
while ($row = $stmt->fetch(PDO::FETCH_OBJ)) {
        echo "nazwisko {$row->NAZWISKO}, etat {$row->ETAT}<br>"; }
?>
```

      - PDO::FETCH_LAZY – kombinacja PDO::BOTH oraz PDO::FETCH_OBJ, własności obiektu są tworzone w momencie odwołania się do nich
      - PDO::FETCH_BOUND – zwraca TRUE, jeżeli powiodło się pobranie kolejnego wiersza do zmiennych związanych z kolumnami zbioru wynikowego (patrz: PDOStatement->bindColumn)
- **PDO zwraca kolumny zbioru wynikowego jako wartości typu `string`**

Przeglądanie zbioru wynikowego (3)

- **`PDOStatement->bindColumn()`– metoda wiążąca zmienne z kolumnami zbioru wynikowego**
  - **adresat:**
    - obiekt klasy `PDOStatement` reprezentujący przygotowane i wykonane polecenie
  - **parametry:**
    - numer pozycji (od 1) lub nazwa kolumny zbioru wynikowego
    - zmienna wiązana z kolumną zbioru wynikowego
    - typ danych zmiennej wiązania (opcja)
  - **wyjście:**
    - TRUE w przypadku sukcesu, FALSE w przypadku porażki

```
$stmt->bindColumn('NAZWISKO', $nazwisko);
$stmt->bindColumn('ETAT', $etat);
while ($stmt->fetch(PDO::FETCH_BOUND)) {
     echo "Nazwisko $nazwisko, etat $etat\n";
}
```

Kursor przewijany (1)

- **`PDOStatement->fetch()`– metoda pobierająca wiersz zbioru wynikowego**
  - **adresat:**
    - obiekt klasy `PDOStatement` reprezentujący przygotowane i wykonane polecenie
  - **parametry:**
    - stała określająca typ wartości zwrotnej
    - kierunek pobierania:
      - PDO::FETCH_ORI_NEXT – pobiera następny wiersz
      - PDO::FETCH_ORI_PRIOR – pobiera poprzedni wiersz
      - PDO::FETCH_ORI_FIRST – pobiera pierwszy wiersz
      - PDO::FETCH_ORI_LAST– pobiera ostatni wiersz
      - PDO::FETCH_ORI_ABS – pobiera wiersz wskazany za pomocą bezwzględnego przesunięcia pobierania
      - PDO::FETCH_ORI_REL – pobiera wiersz wskazany za pomocą przesunięcia pobierania względem bieżącej pozycji kursora
    - przesunięcie pobierania (opcja)
  - **wyjście:**
    - wartość zwrotna, której typ jest zależny od wartości 1-szego parametru (analogicznie jak w przypadku operacji pobrania rekordu dla kursora zwykłego)

Kursor przewijany (2)

```
$sql = 'SELECT nazwisko, etat FROM pracownicy ORDER BY placa_pod';
 try {
   $stmt = $dbh->prepare($sql, array(PDO::ATTR_CURSOR =>  PDO::CURSOR_SCROLL));
   $stmt->execute();
   $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_LAST);
   do {
     $data = $row[0] . "\t" . $row[1] . "\n";
     print $data;
   } while ($row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_PRIOR));
   $stmt = null;
 }
 catch (PDOException $e) {
   print $e->getMessage();
 }
}
```


Modyfikowanie danych

- **wykonanie metody `PDO->exec()` lub `PDOStatement->execute()` i następnie `PDOStatement->rowCount()`**

```
...
$rowCount = $dbh->exec("update pracownicy set placa_pod=placa_pod+1");
echo $rowCount;
...
```

```
...
$stmt = $dbh->prepare("update pracownicy set placa_pod=placa_pod+1");
$stmt->execute();
echo $stmt->rowCount();
...
```


Obsługa transakcji

- **tryby zatwierdzania transakcji:**
  - **automatyczny – każde polecenie jest osobną transakcją, zatwierdzaną zaraz po zakończeniu wykonywania, tryb domyślny,**
  - **ręczny – transakcja musi zostać jawnie rozpoczęta przez wywołanie metody `PDO->beginTransaction()` i zakończona przez wywołanie metody `PDO->commit()` lub `PDO->rollBack()`, brak zakończenia transakcji przed zakończeniem wykonywania skryptu powoduje jej automatyczne wycofanie**

```
...
$dbh->beginTransaction();
$stmt = $dbh->prepare("update pracownicy set placa_pod=placa_pod+1");
$stmt->execute();
echo $stmt->rowCount();
$dbh->commit();
...
```